South African Journal of Information Management
[DoE accredited]

Peer Reviewed Article

Vol.3(2) September 2001

Automatic extraction and analysis of financial data from the EDGAR database

Christoph Leinemann
schlottmann@aifb.uni-karlsruhe.de
Institute AIFB, University of Karlsruhe (TH)

Frank Schlottmann
schlottmann@aifb.uni-karlsruhe.de
Institute AIFB, University of Karlsruhe (TH)

Detlef Seese
seese@aifb.uni-karlsruhe.de
Institute AIFB, University of Karlsruhe (TH)

Thomas Stuempert
stuempert@aifb.uni-karlsruhe.de
Institute AIFB, University of Karlsruhe (TH)

Contents

---

1. Introduction

Contemporary financial markets cause a growing need for quick access to information supporting trading decisions. Since the amount of information available on the World-Wide Web is exponentially growing, it is a huge problem to find useful and reliable information efficiently and at the right time. Intelligent systems (e.g. Almeida Ribeiro *et al.* 1999; Goonatilaki and Treleaven 1995; Frick *et al.* 1996; Hermann et al. 1998) and the new technology of software agents (Klusch 1999) can be useful tools for accomplishing this task.

We follow the latter approach by implementing Edgar2xml, a software agent which extracts fundamental company data from the Electronic Data Gathering, Analysis and Retrieval (EDGAR) database of the United States Securities and Exchange Commission (SEC) and outputs this data in a format which is useful to support stock market trading decisions. The SEC is a regulatory authority for securities markets in the United States with the task to protect investors and to maintain fair, honest and efficient markets. Registered companies are required to file certain financial company data on forms (e.g. 'form 10-Q' for quarterly financial reports and 'form 10-K' for annual reports), which are then made publically available on the EDGAR database. The database can be accessed via a WWW interface and contains documents reaching back to 1994. According to their high relevance for investor decisions, we will concentrate on data extraction from form 10-K filings, although the methods presented in this article can be applied to any other type of filing as well.

While companies use some SGML tags in their filings, documents in the EDGAR database generally contain only very few tags, for example <TABLE> at the beginning and </TABLE> at the end of balance sheet information on some but not all 10-K forms. The balance sheet itself is pure ASCII text and can easily exceed a size of 200 KB. There is a need for automatic extraction of relevant data, because investors who are interested in quantitative balance sheet information naturally prefer immediate online access to relevant data over having to read the entire 10-K filing.

Currently several EDGAR agents exist, for example EDGAR online, 10-K Wizard and FRAANK(Kogan *et al*. 1998). They are used on portal sites for general financial information (e.g. at Yahoo! and BigCharts). These EDGAR agents, however, do not fragment a balance sheet into its components but can only extract well structured passages, that is, extract a whole balance sheet. They are unable to extract detailed information like single balance sheet items from the ASCII passages.

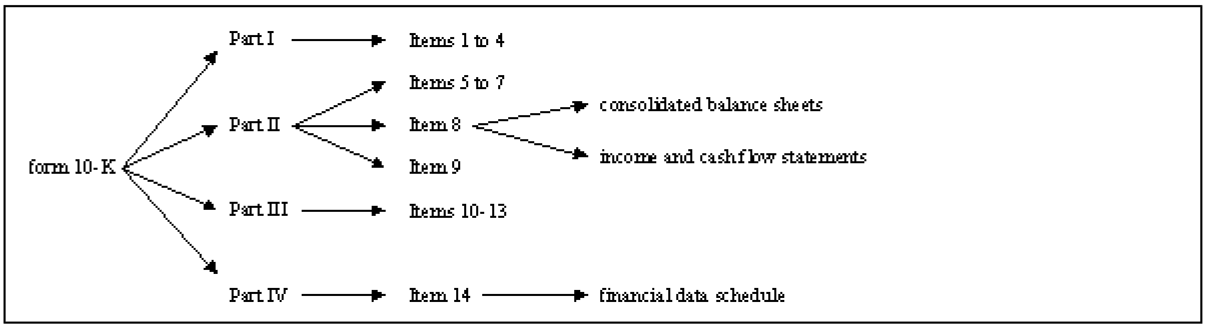
If not only the online extraction of entire balance sheets but also the process of online financial analysis is supported by software, investors will be able to analyse companies faster and more conveniently. This is the goal of our software agent, Edgar2xml, which will be introduced in the following sections.

top

2. Structure of SEC 10-K filings containing semistructured company data

Automatic extraction of unstructured information is an almost impossible task. Hence our agent uses the semistructure found in some SEC 10-K filings. Valid for a specific company and a single year, form 10-K is divided into four parts. Each part consists of several different sections of the annual report of the company which are organized as numbered items. An overview of form 10-K's structure with special respect to those items containing balance sheets and other financial information is given below (Skousen 1991; Leinemann 2000).

Figure 1 Overview of form 10-K structure



We focused our work on items 8 and 14, which contain audited balance sheets for two years as well as three audited annual statements of income and cash flow, and supplemental financial data schedules (FDS). The FDS consist of aggregated financial information and selected financial ratios.

Only the FDS are sufficiently tagged to be machine understandable in the sense of XML. Our software agent, however, implements a new Java-based methodology for mining even other, semistructured parts of a balance sheet.

We cannot query SEC filings in a database-like fashion based on their underlying structure. However, we can provide database-like querying for semi-structured sources by building wrappers around these sources. Each source is wrapped with a translator (or wrapper) that logically converts the underlying data objects into a common information format. Before we will be able to introduce Edgar2xml agent it is therefore necessary to have a closer look at dextrapi wrapper, a general API for Data EXTRaction.

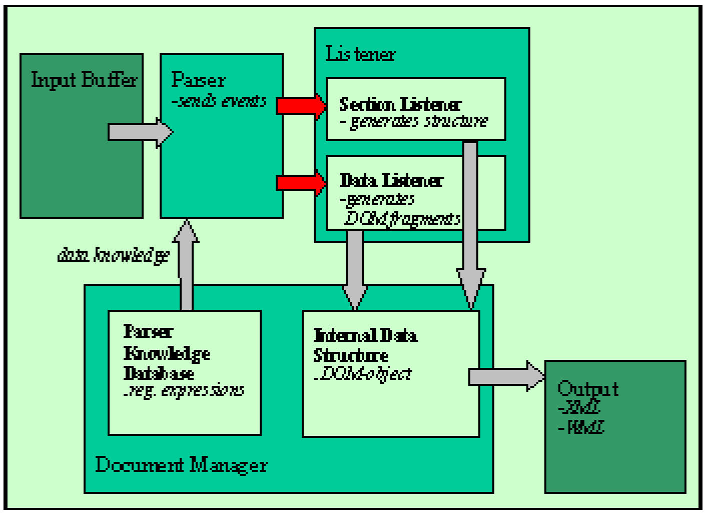
top

3. Dextrapi wrapper

3.1 Overview

The dextrapi wrapper is a framework for extracting information, for example financial data, from any text-based resource. The extracted information is transformed into XML syntax. Most wrapper approaches (Azavant and Sahuguet 1999; Huck *et al.* 1998) need a fixed structure, for example HTML structure, in the file they have to extract from. Dextrapi is able to process even ASCII text sources with changing structure, that is dextrapi detects the beginning of a balance sheet whether the beginning is marked with a tag or special ASCII text. The following picture shows the architecture of dextrapi:

Figure 2 Dextrapi architecture



The data extraction process is accomplished as follows:

1. Text from external sources is written into an input buffer.

2. The text in the input buffer is read by a parser.

3. The knowledge database of the document manager provides the parser with regular expressions . These regular expressions are needed for section and keyword identification. We define section identification as identification of the position where the extraction process starts or ends, for example the beginning of some table. Keyword identification detects items which should be extracted. The document manager organizes the parsing process.

4. For section identification, the parser fires an event if a regular expression in the parser's knowledge base matches some ASCII text in the input buffer. This event activates the data listener.

5. The data listener is activated on receiving an event from the parser. Extraction within a document's section is done by the parser. The parser reads out the input buffer until the data listener detects a structure specified by a regular expression. If the specified structure is detected, relevant information has been found and the parser's data event class fires an event to the listener.

6. Each detected keyword is transformed to a Document Object Model (DOM) element by the listener, that is, the content of a section is transformed to be part of a DOM tree, elements of this DOM tree are generated and sent to the document manager.

7. The internal data structure of the document manager generates a DOM tree from the DOM fragments of the data listener. Each DOM element represents relevant information that is specified by regular expressions and that is to be extracted. The section listener generates a representation of the text document's structure which is then passed to the internal data structure of the document manager.

8. The document manager writes elements of the DOM tree to an XML output stream which conforms to an XML schema.

3.2 Parser

The parser gets its information from the knowledge database that describes when to fire an event. We define parser knowledge as a set of keywords and regular expressions that the parser needs to fire events if an ASCII text matches some element of this set. Parser knowledge consists of section knowledge and data knowledge. The section knowledge for identification of the sections (e.g. SEC Header for company identification, balance sheet, FDS) has the columns as shown in Table 1.

Table 1 Section knowledge

Name	Name of a section
sectionID	key for identification of the section
section-Beginning	contains a regular expression matching the beginning of a section
sectionEnding	contains a regular expression matching the ending of a section
subSectionOf	contains the sectionID of its parent section or '-1' if it is the root section. There can only be one root section
beginning-Scope	can have the value 'cursor' or 'buffer' and relates to the scope which is subject to the match of the beginning regular expression
endingScope	can have the value 'cursor' or 'buffer' and relates to the scope which is subject to the match of the regular ending expression

burnable	can have the value true or false. If true the section occurs only once per encapsulating section. If false the section can occur several times within it's parent section

We use section knowledge and data knowledge to optimise the parsing process, which is organized in two steps. In the first step, interesting sections (SEC-Header, a whole balance sheet, the Financial Data Schedules) are identified. In the second step, data knowledge is used for keyword detection, identifying the data (financial items) in a section that is to be extracted. The separation of structural information, like the beginning of a section, from data extraction information (e.g. keywords detecting a certain structure in the underlying ASCII text) results in an efficient parsing process.

The data knowledge consists of knowledge about the data in a section that is to be extracted and has the columns as shown in Table 2.

Table 2 Data knowledge

Name	**Name of the data to be extracted**
dataId	serving as a primary key
data	a regular expression matching the data associated with the name
parent	a sectionID within this data occurs. This is a foreign key and relates to the field 'sectionID' in the section knowledge entity
priority	an integer determining the order in which the events are fired in case they occur in the same scope
scope	this field is the same as the scope of the section knowledge entity
depth	the number of bracket pairs the data of the match is embraced in
burnable	can have the value true or false. If true the data occurs only once per encapsulating section. If false the data can occur several times within its parent section

The parser reads the SEC filing through a buffer that consists of several lines. Within the buffer one line represents the cursor. While the buffer moves through the file, the buffer fires two kind of events, section events and data events. The list of section events which are fired is dynamically changing according to the section knowledge from the knowledge database. For every section that is entered, the possible subsection events and the section ending event are loaded into the firing list. The firing list for data events is changed with the parser's plugKnowledge(String section) method, which queries the database for the data events that occur in a section and adds them to the firing list. This is useful when only data of a certain section is of interest, like a balance sheet section, because it avoids redundant data events to be fired throughout the whole document.

The parser registers and unregisters data event listeners and section event listeners. When dealing with section events, the parser performs event-single-casting, that is that only one listener object can register to the parser. Concerning data events the parser serves as an event-multi-caster, that is several data listener objects can register to receive data events. The document manager registers itself as a section listener and manages which regular expressions of its knowledge database (see Figure 2) are activated and therefore sent to the listener registry. The registry administers these regular expressions, if a regular expression is not needed any more, it will be deleted from the registry. One or several data listener objects can register to map data. The firing process is called every time when a new line is added to the buffer. According to the regular expressions that have matched buffer and cursor some events are instantiated and methods of the registered listeners are called passing the appropriate event as a parameter.

3.3 Listener

The listener interface requires the methods data(DataEvent). The data() method is supplied with a formal parameter of type DataEvent. The DataEvent object encapsulates two objects. One is the match-object encapsulating the data that has been retrieved and the other one is a string-object representing the name of the data. The listener stores information in its internal DOM representation.

The data listener returns a DOM element representing the document fragment that the data listener has created. The data event object encapsulates two objects of type match and string. The match object encapsulates the data that has been retrieved. It can be accessed through its toString() method. Invoking the getName() method on data event returns a string representing the name of the data. The data will be stored in an appropriate place within an internal DOM element, for example by appending a DOM text node that includes the data or by appending an attribute name-value-pair or by creating a sub-element and then appending a text node. A mapping to an internal DOM structure can be performed. Invoking the getData() method on data listener returns the internal DOM element that has been modified through data method invocations. This element now contains a document fragment that can be further processed from the document manager that has invoked getData().

The section listener returns a DOM object that represents the whole document that has been parsed. Two more methods, sectionBeginning(SectionEvent) and sectionEnding (SectionEvent) are required, they are invoked when sections as defined in section knowledge begin and end.

3.4 Document manager

The document manager consists of section listeners and holds a reference on the parser creating data listener objects if the document manager gets an event for extracting a section. The sectionBeginning() method supplies the parser with a certain class of events to fire and registers the matching data listener object. This class of events can be queried from the parser knowledge database according to the current section.

The number of events the parser fires and the number of listeners that are registered is crucial to the performance of the parser. To achieve high performance the sectionEnding() method removes data knowledge from the parser by deleting the class of events associated with the ending section from the parser's firing list and data listener objects will be set unregistered. This improves the speed of the parsing process because the number of target methods which are called while firing an event are reduced.

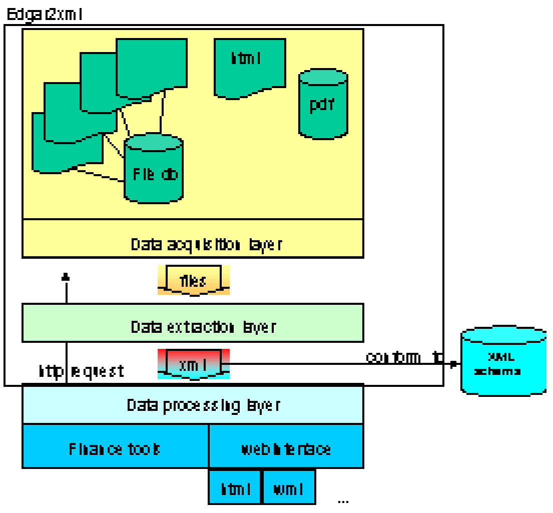
Now we are able to describe Edgar2xml, the agent for extracting financial data from SEC filings.

top

4. Edgar2xml

First we give an overview of the Edgar2xml agent architecture and explain the components, that is, the different layers (see **Figure 3**).

Figure 3 Overview of Edgar2xml's architecture



4.1 Data acquisition layer

The data acquisition layer is responsible for retrieving the target file and for opening a data input stream that is passed to the data extraction layer, that is it extracts the text-based data from the EDGAR database and writes it to the input buffer. The data acquisition layer receives a request with the parameters company identification and year from the processing layer and the data acquisition layer sends the text-based data to the data extraction layer.

4.2 Data extraction layer

The data extraction layer performs the text mining process. It receives the data in its proprietary format and processes it. A parser reads the data input stream and fires corresponding events identifying relevant data that hane been found. The parser reads regular expressions from the knowledge database.

In Figure 4 a typical sample input file is presented showing the architecture of a 10-K filing. Keywords for detecting the balance sheet section are in this case 'balance', 'sheet', '<table>' and 'Current assets'. In this example there are only two listed balance sheet items, namely 'Cash and cash equivalents' and 'Short-term investments'.

Figure 4 Excerpt from form 10-K for Intel Corp.

```
<SEC-DOCUMENT>0001012870-00-001562-index.html : 20000324

<SEC-HEADER>0001012870-00-001562.hdr.sgml : 20000324

ACCESSION NUMBER:   0001012870-00-001562

CONFORMED SUBMISSION TYPE:      10-K

PUBLIC DOCUMENT COUNT:          6

CONFORMED PERIOD OF REPORT:     19991225

FILED AS OF DATE:   20000323
```

```
FILER:

   COMPANY DATA:

   COMPANY CONFORMED NAME:     INTEL CORP

   CENTRAL INDEX KEY:     0000050863

   STANDARD INDUSTRIAL CLASSIFICATION:     SEMICONDUCTORS & RELATED DEVICES
[3674]

   IRS NUMBER:941672743

.

.

Page 14

<PAGE>

       Consolidated balance sheets

December 25, 1999 and December 26, 1998

(In millions--except per share amounts)

<TABLE>

<CAPTION>

1999                    1998

--------    --------

<S>                                                        <C>         <C>

Assets

Current assets:

  Cash and cash equivalents                          $ 3,695    $ 2,038

  Short-term investments                               7,705       5,272

.

.(..... items of a balance sheet)

. Liabilities

.

.(.... items of a balance sheet)
```

The task of the listeners is the following:

Listeners get the events and generate an internal object representation of the events they receive. This object representation of the data can be accessed by the data processing layer. The data extraction layer applies both object-model and event-driven parsing: We use the event driven concept for the identification of the different sections, that is items of a form 10-K and the object model approach, that is a DOM parser, for the representation of the data extracted from these sections. This combination guarantees good performance without a lack of power in rendering the data.

4.3 Data processing layer

The extraction layers store balance sheet items in a DOM object. Now the data processing layer transforms the DOM elements into an XML data output stream. Figure 5 displays such an XML output which contains each detected financial item. Balance sheet items have been converted to XML. This XML data conform to an XML schema, XML Data Reduced (XDR), that is registered at Biztalk, a public repository for XML schemata, that is a set of rules for describing the underlying document structure of the XML document.

Figure 5 Excerpt from an XML output for form 10-K of Intel Corp.

```
<?xml version='1.0'?>

<SECFiling>

<SECHeader>

   <CompanyData>

     <SIC>

       <SICName>SEMICONDUCTORS &amp;
RELATED DEVICES </SICName>

     </SIC>

<CompanyConformedName>INTEL CORP

</CompanyConformedName>

     <IRS>941672743</IRS>

   </CompanyData>

</SECHeader>

<BalanceSheet>

   <BalanceSheetHeader>

     <columns>

       <year>1999</year>

       <year>1998</year>

     </columns>

     <Multiplier>1000000</Multiplier>

     <Currency>USD</Currency>

   </BalanceSheetHeader>
```

```
<AssetsFixed>

        <constructionInProgress>

          <number year='1999'>1460</number>

          <number year='1998'>1622</number>

        </constructionInProgress>

      </AssetsFixed>

</BalanceSheet>



<EX_27>

.... <EX-27Header>

      <Legend> THIS SCHEDULE CONTAINS
SUMMARY

INFORMATION EXTRACTED FROM INTEL
CORPORATION'S

CONSOLIDATED STATEMENTS OF INCOME
AND

CONSOLIDATED BALANCE SHEETS

</Legend>

      <Multiplier>1000000</Multiplier>

      <PeriodType>12</PeriodType>

      <FiscalYearEnd>DEC-25-
1999</FiscalYearEnd>

      <PeriodEnd>DEC-25-1999</PeriodEnd>

    </EX-27Header>
```

```
<Assets>
  <AssetsCurrent>
    <CashAndEquivalents>
      <number year='1999'>3695</number>
      <number year='1998'>2038</number>
    </CashAndEquivalents>
  </AssetsCurrent>
```

```
  <Assets TotalAssets='43849'>
    <CurrentAssets TotalCurrentAssets='17819'>
      <Cash>3695</Cash>
      <Securities>8093</Securities>
</EX_27>
</SECFiling
```

Several recommendations exist at the World-Wide Web Consortium for XML schemas, for example XML-Schema, XML-Data, Document Content Description (DCD) and Document Type Definitions (DTDs). Each XML schema has its own syntax for constructing these rules and each XML schema has a different set of features for defining the rules. This data can be used as a message for other applications like a financial analysis tool that will use XML data for calculating ratios etc. Document Type Definitions do not allow to define data types like integer or real numbers which are necessary for processing financial data. Therefore we use the XDR schema, in which data types for each financial ratio can be defined. For presentation purposes the XML data are transformed by an XSL-processor into formats like HTML or WML.

top

5. Ongoing work

The activities of building agents for the automatic transformation of fundamental company data into a representation that enables and supports quick trading decisions cannot be separated from standardization activities. In September 1999 the XFRML workgroup was founded from AICPA with the aim to develop a standardized computer language for describing financial reports with only one XML vocabulary. The XFRML specification, a XML dialect for financial applications named XBRL (Extensible Business Reporting Language), is still in use. At the moment XFRML shows only a sample 10-K filing which adapts XML tags to each ASCII-10-K output. Therefore there is a strong need for only one specification for all 10-K filings which enable quick access to 10-K filings supporting financial analysis and fast trading decisions. We have shown that it is possible to automatically extract financial data and transform the data into XML. With the usage of XML for describing financial data, the data become machine understandable and reusable. At the moment we do not detect all balance sheet items. Therefore the scope of Edgar2xml can be extended in two dimensions:

- Improving extraction quality: Include more financial items in a single balance sheet and detect synonyms (financial items of different types but with the same meaning). This task could be done by modelling financial data with an ontology.
- Extending extraction scope: Extract not only balance sheet information but also a consolidated statement of income and consolidated statement of cash flow.

Up to now Edgar2xml has been capable of extracting the financial data of balance sheets and the financial data schedules. By specifying new data events in the database and by implementing new listeners the functionality can be extended to other applications.

Acknowledgement

The authors would like to thank Tobias Dietrich for reading the manuscript and for many helpful suggestions.

top

6. References

Almeida Ribeiro, R., Zimmermann, H.-J., Yager, R. and Kacprzyk, J. 1999. *Soft computing in financial engineering, studies in fuzziness and soft computing.* Heidelberg: Springer.

Azavant, F. and Sahuguet, A. 1999. Web ecology: recycling HTML pages as XML documents using W4F. Submitted to WebDB'99, Philadelphia, June 1999.

Goonatilake, S. and Treleaven, P. (eds.) 1995. *Intelligent systems for finance and business*. Chichester: Wiley.

Frick, A., Herrmann, R., Kreidler, M., Narr, A. and Seese, D. 1996. A genetic-based approach for the derivation of trading strategies on the German stock market. In Amari, S.; Xu, L; Chan, L.-W.; King, I. and Leung, K.-S. (eds.) *Proceedings of the 3rd International Conference on Neural Information Processing, Hong Kong, September 1996*. Singapore: Springer, 1996, pp. 766-770.

Herrmann, R., Kreidler, M., Seese, D. and Zabel, K. 1998. A fuzzy-hybrid approach to stock trading. In Usui, S. and Omori, T. (eds.) *Proceedings of ICONIP'98-Kitakyushu, The Fifth International Conference on Neural Information Processing, Kitakyushu, Japan, October 1998*. Amsterdam: IOS Press, 1998, pp. 1028-1032.

Huck, G., Fankhauser, P.,Aberer, K. and Neuhold, E. 1998. JEDI: Extracting and synthesizing information from the Web. Submitted to COOPIS 98, New York: IEEE Computer Society Press.

Klusch, M. 1999. *Intelligent information agents - agent-based information discovery and management on the Internet*. Springer.

Kogan, A., Nelson, K., Srivastava, R., Vasarhelyi, M. and Lu, H. 1998. FRAANK: Financial reporting and auditing agent with Net Knowledge. Collected abstracts of the American Accounting Association Annual Meeting, 1998.

Leinemann, C. 2000. Examination of public companies' financial information dissemination in Europe and the USA. Proposal and sample implementation of a software agent for the use of extracting and disseminating financial information for the purpose of maximal reusability, diploma thesis, Institute AIFB, University of Karlsruhe (TH).

Skousen, F. 1991. *An introduction to the SEC*. Cincinnati: South-Western.

top

he/she/they may have or acquire against the publisher, its suppliers, licensees and sub licensees and indemnifies all said persons from any claims, lawsuits, proceedings, costs, special, incidental, consequential or indirect damages, including damages for loss of profits, loss of business or downtime arising out of or relating to the user's use of the Website.

top

ISSN 1560-683X
Published by InterWord Communications for the Centre for Research in Web-based Applications, Rand Afrikaans University